VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PROGRAM PRO GENEROVÁNÍ OBRÁZKOVÝCH KŘÍŽOVEK

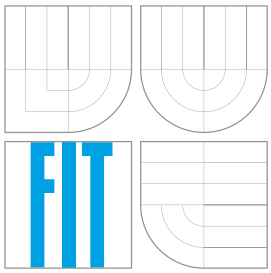
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ONDŘEJ NAVRÁTIL

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PROGRAM PRO GENEROVÁNÍ OBRÁZKOVÝCH KŘÍŽOVEK

PROGRAM FOR GENERATION OF THE IMAGE CROSSWORDS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ONDŘEJ NAVRÁTIL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN KAŠTIL

BRNO 2011

Abstrakt

Ústředním tématem této práce jsou obrázkové křížovky, tedy hlavolamy podobné např. známému sudoku či criss cross. Pozorost je věnována zejména problematice automatického a poloautomatického generování těchto rébusů z digitálních obrázků a otázce řešení, respektive řešitelnosti obrázkových křížovek. Zkoumané vlastnosti a postupy jsou demonstrovány v aplikaci picross, implementované jako součást této bakalářské práce.

Abstract

Main topic of this thesis is the picross, a puzzle similar to well-known sudoku or criss cross. Central questions discussed in the text are automatic and semi-automatic generation of the rebus from digital images and solving and solvability of the puzzle. Characteristics and procedures described are demonstrated in the aplication picross, which was implemented as a part of this bachelor's thesis.

Klíčová slova

Obrázková křížovka, malovaná křížovka, hlavolam, zpracování obrazu, prohledávání stavového prostoru.

Keywords

Picross, nonogram, griddler, puzzle, image processing, state space search.

Citace

Ondřej Navrátil: Program pro generování obrázkových křížovek, bakalářská práce, Brno, FIT VUT v Brně, 2011

Program pro generování obrázkových křížovek

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jana Kaštila. Další informace mi poskytl Ing. Jiří Giesl, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ondřej Navrátil
18. 5. 2011

Poděkování

Za poskytnuté rady a odbornou pomoc děkuji vedoucímu mé práce Ing. Janu Kaštilovi a dlouholetému příteli Ing. Jiřímu Gieslovi, Ph.D.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Oblast hlavolamů zaznamenává v posledních letech celosvětový rozvoj. Příkladem může být stále populárnější sudoku. Nejrůznější druhy lušťovek jsou zajímavou kombinací zábavy a jisté formy učení a trénování mozku, není proto divu, že se těší takové oblibě.

Spolu s rozvojem stávajících a vynalézáním nových hlavolamů se na poli informatiky a matematiky otevírají další disciplíny. Jejich náplň bývá různorodá a pojímá jak algoritmy automatizovaného řešení hlavolamů, tak metody vytváření jejich zadání.

Tato práce se zabývá problematikou obrázkových křížovek, což je druh číselného hlavolamu, jehož řešením je binární rastrový obrázek. Stěžejním cílem je vytvoření programu, který dokáže z digitálního obrázku vygenerovat zadání obrázkové křížovky. Ukážeme však, že k vytvoření kvalitního, vyluštitelného zadání je třeba zabývat se i automatickými metodami řešení obrázkových křížovek.

V kapitole 2 jsou popsány základní vlastnosti křížovek a metody možného řešení. Kapitola 3 se zabývá zpracováním obrazu, konkrétně použitými obrazovými filtry a postupy pro vytvoření křížovky z obecného obrázku. Kapitola 4 se věnuje popisu funkčnosti a implementace aplikace `picross` vzniklé jako součást této práce.

Kapitola 2

Obrázkové křížovky

V této kapitole jsou popsány základní pravidla a definice týkající se obrázkových křížovek, je nastíněno několik metod řešení a problémy s nimi svázané. Bude diskutováno též užití těchto metod při počítačovém řešení různých druhů úloh a možnosti dalších rozšíření.

2.1 Historie

Obrázkové křížovky jsou příbuzné hlavolamům jako kris-kros či klasickým křížovkám. Jejich vznik popisuje na svém webu James Dalgety [3].

První krok můžeme datovat do roku 1987, kdy japonský grafik Non Ishida vytvořil obraz pomocí rozsvěcování a zhasínání světel na mrakodrapu. Ideu obrazu ve formě čtvercové sítě poté aplikoval do série hlavolamů nazvaných „Window Art Puzzles“, publikovaných r. 1988 v Japonsku. Jednalo se o úplně první hlavolam tohoto typu, který byl kdy publikován.

Roku 1989 představil Non Ishida svůj rébus anglickému milovníku hlavolamů Jamesi Dalgetymu. Dohodli se na spolupráci a r. 1990 začaly obrázkové křížovky vycházet v nedělní příloze anglického deníku *The Telegraph* pod názvem *Nonogram* (z „Non“ Ishida a Dia„gram“).

V posledních letech se obrázkové křížovky dostaly na obrazovky herních zařízení jako Game Boy, Nintendo DS atp. Hlavolam je znám pod několika názvy – *Picross* (z angl. „Pic“ture „Cross“word, obrázková křížovka), Griddler (z angl. „Grid“, mřížka). Pravidla ani způsob řešení se však nijak nezměnila. Příkladem může být hra Picross DS [8] z r. 2007.

V současnosti existuje mnoho různých jednotlivců a sdružení zabývajících se obrázkovými křížovkami. Známá je mezinárodní komunita kolem webových stránek griddlers.net [17]. Vyvíjejí se též různé mutace křížovek, jako např. triddlers [17], trojrozměrné křížovky [7] a další. U nás vychází například luštitelský časopis Malované křížovky [14].

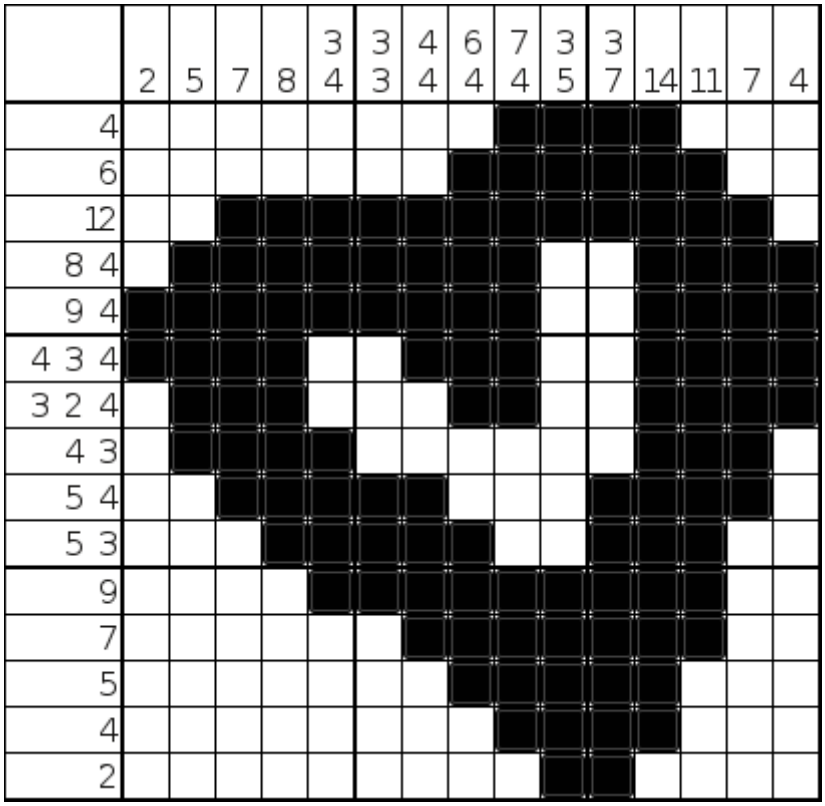
2.2 Zadání

Zadání konkrétní křížovky je dáno čtvercovou sítí s rozměrem $m \times n$ (vydavatelé používají pro m, n hodnoty dělitelné pěti, aby řešitelům usnadnili počítání) a ohodnocením jednotlivých řádků a sloupců ve formě uspořádané n-tice celých čísel.

Úkolem luštitele je vybarvit jednotlivá pole v souladu s ohodnocením sloupců a řádků tak, aby ve výsledku vznik černobílý rastrový obrázek, jak je vidět v obrázku 2.1.

Řešení je úzce spjato s ohodnocením. Ohodnocení, reprezentované čísly u řádků a sloupců,

Obrázek 2.1: Příklad obrázkové křížovky

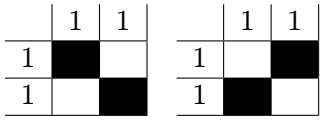


vyjadřuje množství a délku[1] navzájem disjunktních spojitých vybarvených částí řádku či sloupce, které jsou odděleny jedním či více nevybarvenými poli. Uspořádání čísel je shodné s uspořádáním odpovídajících bloků a zobrazení mezi čísly a bloky je bijektivní.

2.3 Jednoznačnost zadání

Významným problémem tohoto typu úloh je fakt, že zobrazení množiny řešení na množinu jejich zadání není injektivní – existují tedy taková zadání, která vedou k více řešením. Příkladem budiž dvojice různých křížovek se shodným ohodnocením sloupců a řádků na obrázku 2.2.

Obrázek 2.2: Příklad nejednoznačného zadání



Naopak z determinističnosti postupu pro ohodnocení řádků a sloupců vyplývá, že každému řešení odpovídá právě jedno zadání.

[1] ve smyslu počtu polí

2.4 Detekce nejednoznačnosti

Jak jsme si ukázali v sekci 2.3, existují zadání, která nelze jednoznačně vyřešit. Při generování křížovky je třeba tyto situace identifikovat, jelikož uživatel nemá zájem na vytvoření nevyluštitelné křížovky. V této sekci jsou popsány metody použité pro detekci nejednoznačnosti.

2.4.1 Hledání obecných vzorů

Při bližším zkoumání problému lze vypozorovat několik situací, které jsou příčinou nejednoznačnosti zadání. Často se jedná jakousi diagonální symetrii dvou shodných útvarů v protějších rozích pomyslného obdélníku, jehož zbývající rohy jsou prázdné. V takovémto případě nelze jen podle zadaného ohodnocení jednoznačně určit, kde přesně se tyto útvary budou nacházet, jak vysvětluje obrázek 2.3.

Obrázek 2.3: Obecný vzor nejednoznačnosti

a_1	$a_1{}'$
$a_2{}'$	a_2

Bohužel toto není zdaleka jediný případ vzniku nejednoznačnosti zadání. Při hlubší analýze byla účelově vytvořeným programem využívajícím algoritmus 1 vygenerována sada řešení křížovek s určitými rozměry a z nich vybrány skupiny, jejichž zadání se shodují. Nehledě na množství takovýchto dvojic (a obecně N-tic) kolizních řešení se nepodařilo vzájemné vztahy jakkoli generalizovat. Celý problém se ukázal mnohem komplikovanější, než se prve zdálo, a hledání obecných nejednoznačných vzorů za účelem identifikace neřešitelnosti bylo zavrženo. Matematické analýzy vzniku nejednoznačností, případně jejich detekce, by se však mohly stát zajímavou oblastí pro další bádání a budoucí rozšíření programu.

2.4.2 Řešení křížovky

Dalším možným přístupem k problému kolizních řešení je nechat program, aby se pokusil křížovku vyluštit. Pokud při tom narazí na překážku, je zřejmé, že ani uživatel nebude moci tuto křížovku vyluštit až do konce. Nalezení algoritmu pro luštění křížovky je tedy vhodným řešením problému nejednoznačnosti zadání. Jednotlivé algoritmy jsou popsány v následující sekci.

2.5 Metody řešení

Ukážeme si celkem tři metody řešení křížovek. Vstupem všech tří je zadání křížovky, jejich výsledky se však liší. Každá má své výhody a nevýhody, které budou popsány v následujícím textu.

2.5.1 Kategorizace problému

Lze dokázat, že řešení obrázkové křížovky spadá do kategorie NP-úplných úloh (angl. *NP-complete*, důkaz viz [6]). Význačnou vlastností těchto úloh je fakt, že neznáme úplný [19] algoritmus, který by byl schopen je vyřešit s polynomiální složitostí [5]. Nelze však s jistotou

tvrdit, že takový algoritmus nelze sestavit, jelikož se jeho existenci zatím nepodařilo nijak vyvrátit. Toto tvrzení, obvykle prezentované jako $P = NP$, bylo dokonce zařazeno mezi tzv. Sedm problémů tisíciletí [2]. Hledání úplného algoritmu pro řešení křížovky s jinou nežli exponencionální složitostí je mimo rozsah této práce a v dalším textu budeme uvažovat, že takovéto řešení neexistuje.

Pro účely kontroly jednoznačnosti řešení by intuitivně nemuselo být potřebné křížovku řešit. Stačilo by nám pouze pokusit se nějakým způsobem najít na základě současného ještě další řešení odpovídající zadání. Jak však dokázali Ueda a Nagao [6], také tento problém, popisovaný jako „problém dalšího řešení“ (*angl. Another Solution Problem, ASP*), spadá do kategorie NP-úplných problémů a řešící algoritmus by měl opět exponenciální složitost.

S jistými omezeními, která v některých metodách řešení provedeme, však lze složitost redukovat na úkor úplnosti algoritmu.

2.5.2 Řešení silou

Nejnaivnější způsob řešení vychází z prohledávání stavového prostoru pro celou křížovku s daným rozměrem neinformovanými metodami BFS/DFS/Backtracking [19]. Postup takovéhoto řešení je patrný z algoritmu 1.

Algoritmus 1: Řešení silou

1. Vygeneruj všechna možná řešení křížovky s daným rozměrem.
2. Z řešení vyber ta, jejichž ohodnocení sloupců a řádku (zadání) se shodují se vstupem.

Je zřejmé, že složitost tohoto algoritmu je v závislosti na rozměrech křížovky $(m \times n)$ exponenciální, jelikož počet možných řešení K je dán vztahem $K = 2^{(m.n)}$ [1]. Snahou tedy bude počet generovaných řešení zmenšit. Budeme vycházet z předpokladu, že generování možných řešení probíhá itertivně či rekurzivně podle algoritmu 2.

Algoritmus 2: Generování možných řešení pro algoritmus 1

1. Vygeneruj prázdnou křížovku, nastav startovní pozici na 0.
2. Vezmi aktuální křížovku, a vygeneruj 2 její kopie, jednu s černým polem na startovní pozici a jednu s bílým polem na startovní pozici.
3. Zjisti, zda-li je startovní pozice rovna konci křížovky (tzn. doplnili jsme poslední prázdné pole):
   (a) Pokud ano, přidej do konečné množiny dvě naposledy vygenerovaná řešení a rekurzi ukonči.
   (b) Pokud ne, inkrementuj startovní pozici a proveď bod 2 s oběma naposledy vygenerovanými řešeními a touto novou hodnotou startovní pozice.

Uvažujeme, že startovní pozice s u křížovky s m řádky a n sloupci funguje jako index řádku i a sloupce j s pomocí operace modulo, tedy $i = s/n,\ j = s \mod n$. Z toho vyplývá,

že řešení jsou generována po řádcích. Do algoritmu plnění jednoho řešení potom můžeme zapojit následující optimalizace:

- *Počítání černých polí.* V kroku 1 algoritmu 2 si spočítej, kolik černých polí se má ve výsledku objevit. To se dá jednoduše zjistit sečtením všech čísel v ohodnocení řádků, případně sloupců. Při plnění volných polí v kroku 2 algoritmu 2 potom kontroluj, zda-li už toto číslo nebylo přesaženo, příp. jestli je ještě dostatek volných polí pro jeho dosažení. Při nesplnění některého z těchto kritérií již současné řešení dále nerozvíjej.

- *Průběžné vyhodnocování.* Pokud je v kroku 2 dokončen řádek, respektive sloupec (tedy pokud platí $j = n$, respektive $i = m$), porovnej jeho ohodnocení s ohodnocením na vstupu. Pakliže je mezi nimi rozdíl, dále toto řešení nerozvíjej.

- *Častější průběžné vyhodnocování.* Tato optimalizace ještě více rozvíjí myšlenku průběžného vyhodnocování. Můžeme kontrolovat např. tyto vlastnosti:

  – Při dokončení bloku (vložení bílého pole za černé pole) zkontroluj, jestli se i s odpovídající délkou nachází v ohodnocení na vstupu.

  – Při prodlužování bloku (vkládání černého pole) zkontroluj, zda-li délka nepřevyšuje ohodnocení na vstupu.

  – Při vkládání bílého pole zkontroluj, zda-li ještě v řádku či sloupci zbývá dost místa pro vložení zbývajících bloků.

Při zapojení optimalizací lze algoritmus popsat jako informovaný. Algoritmus je však stále velice pomalý, pro křížovky s $m \geq 10 \wedge n \geq 10$ generuje i se všemi možnými optimalizacemi obrovské množství stavů (blíže viz [1]). Pro samotné řešení křížovky, jehož důležitou vlastností je rychlost, tedy není nejvhodnější. Pro uživatele není přípustné, aby řešení jedné křížovky trvalo déle než několik vteřin. Této době se algoritmus ani zdaleka nepřibližuje.

Nespornou výhodou je však to, že algoritmus je úplný a najde vždy všechna existující řešení. Je tedy použitelný i v případě, kdy je zadání nejednoznačné a jiné algoritmy selhávají.

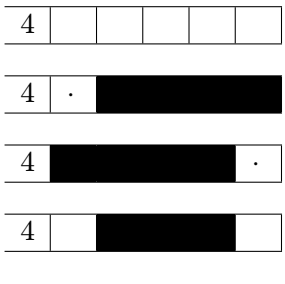
Varianta tohoto algoritmu s první zmíněnou optimalizací byla použita při generování křížovek za účelem nalezení skupin nejednoznačných zadání. Analýza vzorků však, jak již bylo zmíněno v podsekci 2.4.1, nepřinesla žádné hodnotné výsledky.

2.5.3 Uživatelské řešení

Další z metod je inspirována postupy, které užívají lidští řešitelé k vyluštění křížovky. Základním kamenem tohoto způsobu je náhled na jeden konkrétní řádek či sloupec. Ten je potom analyzován za účelem nalezení polí, která budou jistě bílá/černá, ať je již rozestavení bloků jakékoli. Popis těchto metod je uveden například v [1], [17], [3], [14] a obecně je nabízen jako návodný postup ve všech zdrojích publikujících obrázkové křížovky. Metodu lze snadno pochopit z obrázku 2.4. Všechny ostatní metody doplnění jsou založeny na obdobných principech.

S těmito znalostmi tedy můžeme sestavit postup pro vyluštění křížovky obdobným způsobem, jako by to udělal běžný řešitel. Jeden dílčí krok je popsán v algoritmu 3. Jeho principem je prohledávání stavového prostoru pro konkrétní řádek či sloupec a hledání hodnot polí společných pro všechny stavy. Omezením prohledávání na jeden řádek snížíme složitost algoritmu, na druhou stranu tento algoritmus již není úplný.

Obrázek 2.4: Ukázka uživatelského řešení



Algoritmus 3: Uživatelské řešení křížovky

1. Sestroj frontu řádků a sloupců.
2. Vyber řádek či sloupec z fronty. Pokud je fronta již prázdná, tedy v žádném řádku ani sloupci nelze nic jistě doplnit, řešení ukonči jako neúspěšné.
3. Vygeneruj pro něj všechny možnosti doplnění.
4. Hledej průnik všech možností doplnění pro aktuální řádek či sloupec.
   (a) Je-li průnik prázdný, vrať se na krok 2.
   (b) Je-li průnik neprázdný, doplň odpovídající pole do křížovky a řešení ukonči jako úspěšné.

Kroky provádíme tak dlouho, dokud se nám daří vyplňovat nová pole. Pokud je po posledním úspěšném kroku celá křížovka vyplněna, prohlásíme ji za řešitelnou. V opačném případě křížovka nemá triviální řešení.

Algoritmus 3 je velmi vhodné vylepšit zavedením některé z následujících optimalizací:

- *Heuristika výběru řešeného řádku či sloupce.* Při sestrojování fronty v kroku 1, případně při vybírání prvku v kroku 2 se budeme řídit heuristickou funkcí, jejíž hodnoty pro řádky a sloupce odpovídají pravděpodobnosti, že se nám podaří některá jejich pole vyplnit. Lze přitom využít následujících veličin:
  - Délka řádku či sloupce.
  - Délka vektoru ohodnocení.
  - Počet již vyplněných polí.
  - . . . a dalších.

  Pro tuto funkci pochopitelně neexistuje žádný konkrétní předpis a je třeba vycházet z experimentálních poznatků.
- *Potlačení řešení již řešených řádků a sloupců.* Podstata této optimalizace spočívá v úpravě bodu 1. Při generování seznamu vynecháme ty řádky a sloupce, které jsou buď již celé vyřešené, nebo u nich od posledního pokusu o vyřešení neproběhla žádná změna (tedy žádné z jejich polí nebylo vyplněno). První možnost je pochopitelně

podmnožinou té druhé, jelikož ve vyřešených řádcích již žádné změny provést nelze. Opětovné řešení takového řádku či sloupce by nepřineslo žádnou novou informaci[2]. Při implementaci této optimalizace je nutno přiložit ke křížovce také příslušné vektory uchovávající informace o tom, zda se program již pokoušel daný řádek či sloupec řešit. V případě změny některého z polí řešení je třeba nastavit patřičné hodnoty v těchto vektorech na indexech, které odpovídají řádku a sloupci, v nichž se měněné pole nachází. Tato optimalizace přináší značné zrychlení především u složitějších či neřešitelných křížovek. Počet řádků a sloupců zkoumaných při jednom kroku (tedy počet neúspěšných pokusů o řešení) v takovýchto křížovkách je zpravidla vysoký a naopak počet doplněných polí v jednom kroku nízký.

Poněkud komplikovaný je krok 3 řešícího algoritmu. Metoda, jak možnosti řešení vygenerovat, se vzdáleně podobá algoritmu 1. Vhodnými optimalizacemi se však daří dostatečně snížit množství zkoumaných stavů a značně snížit jeho složitost. Postup je popsán v algoritmu 4.

Algoritmus 4: Generování možností řešení řádku či sloupce pro algoritmus 3

1. Nakopíruj současnou podobu řádku či sloupce do vektoru `vzor` a odpovídající ohodnocení do vektoru `ohodnocení`, nastav proměnné `start pole` na 0 a `start ohodnocení` na začátek vektoru ohodnocení.

2. Pokud `start ohodnocení` je za posledním indexem vektoru `ohodnocení` (v předchozím kroku jsme tedy vložili poslední černý blok přítomný v řádku).

   (a) Zkontroluj, zda-li se ve vektoru `vzor` za indexem daným proměnnou `start pole` nenachází žádné černé pole.

   (b) Pokud ano, zahoď současné řešení a ukonči rekurzi.

   (c) Pokud ne, vyplň zbývající dosud neoznačená pole jako bílá a přidej současnou možnost do výsledné množiny.

3. Pokud ve vektoru `vzor` od pozice `start pole` není již dost místa pro vložení zbývajících bloků z vektoru `ohodnocení` včetně mezer mezi nimi, zahoď současné řešení a ukonči rekurzi.

4. Pokus se do vektoru `vzor` vložit bílé pole na pozici dané indexem `start pole`. Pokud to lze, rekurzivně se vrať na bod 2 s takto vytvořeným vektorem jako novou hodnotou proměnné `vzor` a inkrementovanou hodnotou `start pole`.

5. Pokus se do vektoru `vzor` vložit blok černých polí, jehož délka je určena indexem `start ohodnocení` ve vektoru `ohodnocení`, počínaje pozicí `start pole`, včetně následujícího bílého pole (pokud nenásleduje konec řádku). Pokud to lze, rekurzivně se vrať na bod 2 s takto vytvořeným vektorem jako novou hodnotou proměnné `vzor`, inkrementovanou hodnotou `start ohodnocení` a hodnotou `start pole` ukazující za vložený blok (včetně bílého pole).

Algoritmus 4 můžeme opět dále optimalizovat:

[2]Výchozí situace je stejná jako při předchozím pokusu o řešení, vygenerovali bychom tedy shodnou množinu výsledných stavů atd.

- *Sdílení vektoru* `ohodnocení`. Vektor `ohodnocení` se v průběhu generování nijak nemění, je proto možno předávat jej pouze jako ukazatel či deklarovat jako globální proměnnou. Tímto způsobem snížíme nárok na pamět.

- *Sdílení vektoru* `vzor`. Též tento vektor je možno sdílet a nevytvářet pro každé zanoření nový. Před ukončením rekurze je však potřeba navrátit všechny změny, které jsme ve vektoru provedli, abychom nechtěně neovlivnili vyšší úrovně rekurze. Přínosem je opět znatelné snížení nároků na množství alokované paměti.

- *Průběžné porovnávání.* Popisovaný algoritmus je založen na vygenerování všech stavů a následném zkoumání průniku. Stejného cíle, tedy nalezení průniku, však lze docílit i jinak – při vygenerování prvního stavu si jeho vektor zapamatujeme jako `vzorVýsledku`, který bude reprezentovat průnik všech vygenerovaných řešení. Místo přidávání nových řešení do množiny možných výsledků potom pouze porovnáváme nově vygenerované řešení s vektorem `vzorVýsledku`. V místech neshody nastavíme odpovídající pole vektoru `vzorVýsledku` na hodnotu `NEÚSPĚCH`. Poté je možno podniknout následující kroky:

  1. Dostaneme-li se po přidání nového řešení do situace, kdy cely vektor `vzorVýsledku` je naplněn hodnotami `NEÚSPĚCH`, můžeme s generováním dalších řešení přestat. Tato situace totiž naznačuje, že průnik dosavadních řešení je roven prázdné množině. Ze vztahu $X \cap \varnothing = \varnothing$ [18] plyne, že nehledě na počet a podobu dalších řešení budé výsledný průnik prázdný a v tomto řádkuči sloupci se nám nepodaří nic doplnit.
  2. Při vyhodnocování výsledků v algoritmu 3 v kroku 4 poté nemusíme znovu kontrolovat všechna řešení, jelikož jejich průnik i s odpovídajícími hodnotami je zanesen ve vektoru `vzorVýsledku`.

Všimněme si, že tento algoritmus na rozdíl od předchozího nemusí nutně najít řešení. Poskytne nám však cennou informaci ohledně toho, zda-li je pro běžného řešitele možné zadání vyřešit – což je přesně to, co v aplikaci pro generování potřebujeme. Pokud totiž při tomto způsobu řešení narazí program na neřešitelnou situaci, je zřejmé, že i běžný řešitel na tomto bodu zktroskotá.

Zároveň je tento nesrovnatelně rychlejší než první varianta a pro průměrné křížovky (řádově do velikosti 30×30 polí) zabere řešení na běžném stroji méně než vteřinu. Jak si však ukážeme v podsekci 2.5.4, to, že tento algoritmus nedojde do cíle, ještě neznamená, že křížovka nemá jednoznačné řešení.

2.5.4 Víceřádkové řešení

Tento algoritmus je upravenou verzí algoritmu popsaného v podsekci 2.5.3. Jeho název vychází z anglického *Multiline solution*(MLS, viz [17]). Hlavní nevýhodou jeho předchůdce je fakt, že vždy pohlíží pouze na jeden řádek či sloupec. Pokud tento pohled rozšíříme na větší počet takovýchto entit, můžeme dostat více indícií potřebných ke správnému vyplnění křížovky. Není to však tak jednoduché, jak by se mohlo na první pohled zdát. Nejmenší možný počet řádků a sloupců, na které má při takovémto způsobu řešení smysl nahlížet, je čtyři.

Důkaz. V následujícím textu budeme uvažovat pojem entita jako polymorfní označení pro řádek či sloupec. Entitu samotnou potom budeme chápat jako množinu polí tohoto řádku

či sloupce.

Při řešení křížovky pohlížíme vždy na jistou množinu polí E, většinou charakterizovanou sjednocením několika entit $E = \bigcup_{i=1}^{n} e_i$, a snažíme se některá z nich vyplnit. Při vyplňování je třeba potvrdit, že v rámci množiny E neexistuje žádná možnost vyplnění m všech polí z E odpovídající zadání taková, že pole p by mělo jinou barvu, nežli tu, kterou se jej chystáme vyplnit. Teprve poté je možno s jistotou tvrdit, že zvolené pole p bude mít tuto barvu.

Jednořádkový algoritmus pohlíží vždy na jednu entitu $E = e$ a zkoumá všechny stavy m, které mohou nastat. Můžeme tedy říci, že pokud není schopen dokončit řešení, pro každou entitu $E = e$ v křížovce a každé její pole $p \in E$ existují možnosti vyplnění m_1, m_2 takové, že v m_1 je p černé a v m_2 je p bílé.

Nyní se však podívejme na problém řešení z druhé strany. Pokud dokážeme v rámci zkoumané množiny E najít dvě konfigurace m_1, m_2 takové, že v jedné z nich je pole p bílé, a ve druhé černé, je jisté, že pole p při poledu na E vyplnit nedokážeme.

Podotkněme ještě, že rozšířením množiny polí E, na která nahlížíme, nemůže dojít k situaci, kdy bychom pro méně početnou množinu polí nalezli nějakou novou možnost vyplnění. Naopak vlivem většího počtu podmínek, které je pro početnější množinu třeba splnit, lze některé tyto možnosti vyplnění vyloučit. Nemůže tedy nastat situace, kdy bychom pole p byli schopni vyplnit pohlížením na množinu E_1 a naopak nemohli vyplnit pohlížením na množinu E_2, přičemž $p \in E_1, E_1 \subset E_2$.

S těmito vědomostmi můžeme dokázat tvrzení, že pokud selže jednořádkový algoritmus, selže i třířádkový, tudíž pro vyřešení křížovky je třeba pohlížet alespoň na 4 entity.

Vybereme tři entity e_1, e_2, e_3. Množiny možných vyplnění pro $E_1 = e_1, E_2 = e_2, E_3 = e_3$ jednořádkovým algoritmem si označíme M_1, M_2, M_3.

Celkově budeme tedy nahlížet na množinu polí $E_c = e_1 \cup e_2 \cup e_3$.

Nyní dokážeme, že pro kterékoli pole $p \in E_c$ jsme schopni najít obě možnosti celkového vyplnění m_{c1}, m_{c2} množiny E_c, tedy vyplnění všech tří zvolených entit, přičemž v m_{c1} je p černé a v m_{c2} bílé. Možnosti vyplnění m_{c1}, m_{c2} budeme sestavovat z částečných možností vyplnění z M_1, M_2, M_3.

Mohou nastat dvě situace:

1. Všechny zvolené entity jsou rovnoběžné. Vybrali jsme tedy 3 řádky, nebo tři sloupce. Tyto entity nemají žádná společná pole.

   Vezměme libovolné pole $p \in e_1$. Množina M_1 musí obsahovat m_1, m_2 s černým a bílým p. Celková vyplnění m_{c1}, m_{c2} získáme spojením m_1, m_2 s libovolnými možnostmi z M_2, M_3. Nemůže mezi nimi vzniknout žádný spor, jelikož hodnoty jejich polí ani podmínky se nijak neovlivňují.

   Analogicky lze postupovat pro pole v e_2, e_3.

2. Jedna ze zvolených entit je kolmá na ostatní. Vybrali jsme tedy dva řádky a jeden sloupec či naopak. Lze tedy nalézt dvě pole, která jsou společná pro dvě dvojice entit.

   Budeme uvažovat:

   $$p_1 \in e_1 \cap e_2, p_2 \in e_2 \cap e_3 \tag{2.1}$$

   Všechny ostatní případy jsou pouhou analogií s jiným pojmenováním proměnných.

   Pro $p \in e_2$ vezmeme opět m_1, m_2 z M_2. Při výběru možností z M_1, M_3 pro pole v E_1, E_3 je však třeba vybírat ty, které obsahují správnou hodnotu p_1, resp. p_2, určenou v m_1, m_2. Z vyslovených předpokladů však vyplývá, že M_1 i M_3 musí tyto možnosti obsahovat a obě konfigurace m_{c1}, m_{c2} lze proto bez problémů sestavit.

Pro $p \in e_1$ vezmeme m_1, m_2 z M_1. Při výběru možností m'_1, m'_2 z M_2 pro pole v e_2 je třeba znovu vybírat ty, které obsahují správnou hodnotu p_1, určenou v m_1, m_2. Při výběru možností z M_3 pro pole v e_3 nutno vybírat ty, které obsahují správnou hodnotu p_2, určenou v m'_1, m'_2. I zde lze obě konfigurace m_{c1}, m_{c2} bez problémů sestavit.

Analogicky lze postupovat pro pole v e_3.

Dokázali jsme, že pokud selže jednořádkový algoritmus, ani třířádkový nebude schopen žádné z polí doplnit.

□

Metoda víceřádkového řešení je založena na důkazu sporem a je popsána v algoritmu 5. Postup nutný dosažení sporu je ještě lépe patrný z obrázku 2.5.

Algoritmus 5: Víceřádkové řešení

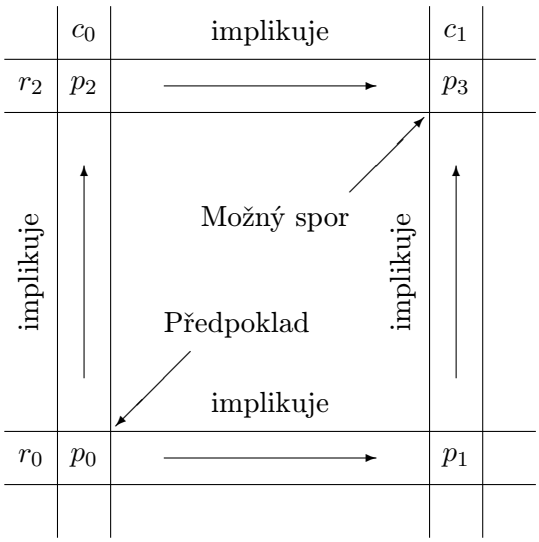
1. Zvolíme předpoklad. Z množiny nevyplněných polí vybereme pole p_0 v řádku r_0 a sloupci c_0 a vyplníme ho.
2. S pomocí předpokladu z bodu 1 se pokusíme algoritmem popsaným v podsekci 2.5.3 vyplnit nějaké pole p_1 v řádku r_0 a nějaké pole p_2 ve sloupci c_0. Pokud se to nepodaří, vrátíme se k bodu 1 a zvolíme jiné pole p_0, příp. jinou hodnotu vyplnění. Máme tedy nyní vyplněna pole $p_0(r_0, c_0), p_1(r_0, c_1), p_2(r_2, c_0)$.
3. Na základě informace o poli p_1 se pokusíme algoritmem popsaným v podsekci 2.5.3 určit hodnotu pole $p_3(r_2, c_1)$.
4. Na základě informace o poli p_2 se pokusíme algoritmem popsaným v podsekci 2.5.3 určit hodnotu pole $p_3(r_2, c_1)$.
5. Porovnáme informace o poli p_3 získané v krocích 3 a 4. Pokud se tyto neshodují, došli jsme ke sporu. Lze říci, že pole p_0 nemůže nabýt hodnotu zvolenou v předpokladu v kroku 1 a vyplníme jej tedy hodnotou opačnou.

Pozn.: Pokud v některém z kroků 3, 4, 5 nedokážeme získat požadovanou informaci, vrátíme se k předchozímu kroku, zvolíme v nich jiná pole a pokračujeme dále. Je také možné dojít ke sporu u jiného pole (podle toho, v jakém pořadí a směru postupné vyplňování provádíme), princip a počet potřebných řádků a sloupců však zůstává shodný.

Nutno podotknout, že je třeba brát opravdu v úvahu minimálně 2 řádky a 2 sloupce, při zkoumání menšího počtu řádků či sloupců bychom dospěli pouze k závěru, že daná kombinace je možná. Bez „zpětné vazby“ dvou implikací v kroku 5 algoritmu 5 však nemůžeme dojít ke sporu a nemůžeme tedy toto řešení vyloučit.

Dokázali jsme, že tento rozšířený algoritmus může nalézt řešení i v případech, kdy jeho triviálnější vzor selže. Na druhou stranu nalezení takovéhoto řešení je velmi náročné i pro počítač disponující velkou výpočetní silou. Požadovat takovéto řešení po běžném uživateli je proto nemožné. Z tohoto důvodu nebyl tento algoritmus implementován a je zde uveden pouze pro kompletnost a jako inspirace pro další rozšíření.

Obrázek 2.5: Grafické znázornění víceřádkového řešení



2.5.5 Odkazy pro další studium

Řešením obrázkových křížovek se zabývají též jiní autoři. V této podsekci je uvedeno několik odkazů na díla jiných řešitelů, která mohou nabídnout alternativní pohled na zkoumanou problematiku. Bohužel, mnoho aplikací s touto tématikou nemá volně dostupné zdrojové kódy či dokumentaci.

Velice zajímavý a vysoce optimalizovaný je program GRID autorů Mirka a Petra Olšákových [11]. Program je psán v jazyce C. Dosahuje vynikajících výsledků z hlediska doby nutné k nalezení řešení, zdrojový kód je dostatečně komentován a dokumentován. Dokáže řešit i složitější varianty a odnože obrázkových křížovek, jako např. vícebarevné křížovky či křížovky s triangulární sítí.

Inspiraci může nabídnout též ročníkový projekt studenta Univerzity Komenského v Bratislavě Mariána Rusnaka [12]. Tento program se též zabývá vytvářením křížovek, avšak nabízí spíše možnost křížovku nakreslit, nežli vygenerovat z obrázku. Aplikace je implementována v jazyce Object Pascal prostředí Delphi, její součástí je i prostý řešící algoritmus. Vývoj tohoto projektu však vzhledem k jeho prezentaci zřejmě nebyl zcela dokončen.

Rozbor problematiky z hlediska umělé inteligence a algoritmů pro prohledávání stavového prostoru nabízí bakalářská práce Lukáše Balcárka [1]. Jsou zde popsány základní metody založené na algoritmech BFS a DFS a také jedna informovaná metoda používající skupinu operátorů simulujících uživatelské postupy řešení.

2.6 Změna hodnoty pole

V této sekci si ukážeme, jaký vliv má změna hodnoty konkrétního pole na řešitelnost, potažmo rozpracované řešení křížovky. Tato informace je pro nás důležitá, jelikož uživatel by měl mít možnost provádět tyto úpravy na vyřešené či rozpracované křížovce. Přitom je nezbytné, aby informace týkající se jednoznačnosti křížovky poskytnutá programem zůstávala validní.

Na jednoduchých případech bude ilustrován vliv změny bílého pole na černé a naopak.

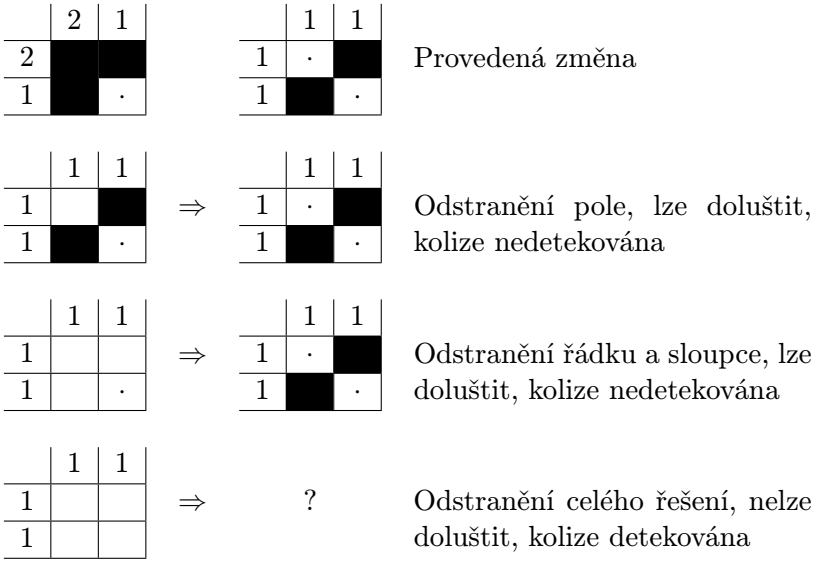
Podstatný je pro nás dopad na rozpracované řešení a určení části, kterou bude třeba revalidovat. Možnosti, které přicházejí v úvahu, jsou zahození pouze řešení daného pole, zahození řešení odpovídajícího řádku a sloupce a zahození řešení celé křížovky.

2.6.1 Změna bílého pole na černé

Tuto situaci ilustruje obrázek 2.6, ve kterém jsme přebarvili pole v prvním řádku a prvním sloupci. Zatímco původní verze křížovky má právě jedno řešení, verze s přebarveným polem jednoznačná není, jak jsme si ukázali v sekci 2.3. Je tedy třeba zjistit, jakou část řešení je třeba vymazat a spočítat znovu, a to tak, abychom odhalili nejednoznačnost.

První možností je vymazání změněného pole. V tomto případě bychom však stále došli k řešení bez jakékoli překážky. Též při odstranění všech polí v souvisejícím řádku a sloupci dojdeme ke stejnému výsledku. Jediným východiskem se tedy zdá vymazat celé rozpracované řešení a pokusit se vyluštit křížovku znovu. Situaci ilustruje obrázek 2.6.

Obrázek 2.6: Příklad změny černého pole na bílé



2.6.2 Změna černého pole na bílé

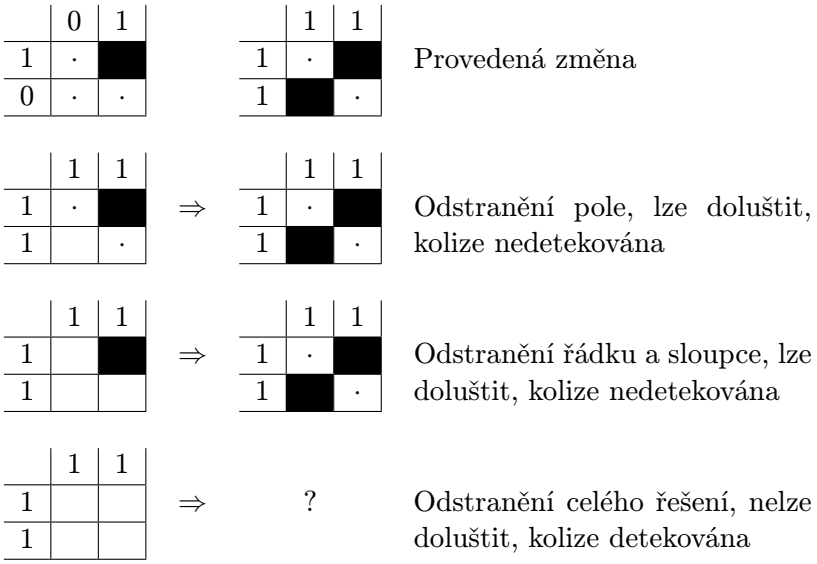
Situaci ilustruje obrázek 2.7, ve kterém jsme přebarvili pole ve druhém řádku a prvním sloupci. Opět jsme se dostali do situace popsané v předchozí podsekci.

Ani v tomto případě nám k identifikaci nejednoznačnosti nepomůže smazání změněného pole či souvisejícího řádku a sloupce a musíme smazat celé dosavadní řešení. Situace je opět znázorněna na 2.7.

2.6.3 Shrnutí

V podsekcích 2.6.1 a 2.6.2 jsme ukázali, že při změně hodnoty kteréhokoli pole je pro identifikaci nejednoznačnosti řešení nutno smazat celé rozpracované řešení. Vyplývá to z faktu, že při luštění nezměněných řádků a sloupců jsme mohli využít již vyluštěného pole v měněném řádku či sloupci. Toto pole však po provedení změny nemusí být vyluštitelné.

Obrázek 2.7: Příklad změny bílého pole na černé



Jedinou možností, jak zachovat alespoň část provedeného řešení, je postup luštění krokovat a pro každý krok si pamatovat řádky a sloupce, které jsme zkoumali. Při změně je poté možno restaurovat všechny kroky provedené od začátku řešení až do kroku, ve kterém jsme zkoumali měněný řádek či sloupec.

Kapitola 3

Generování křížovky

Tato kapitola se zabývá technikami převodu obecného digitálního obrázku na obrázkovou křížovku. Výsledkem tedy má být černobílý rastrový obrázek se zadanými rozměry (zpravidla menšími nežli má zdrojový obrázek).

Mezi podstatné atributy úprav prováděných za účelem získání křížovky lze zařadit

1. Podobu výsledného obrázku (křížovky) s obrázkem na vstupu.
2. Potencionálně velké rozdíly v rozlišení a barevné hloubce mezi výsledným obrázkem (tedy křížovkou) a původním obrázkem.
3. Velice pestrou množinu akceptovatelných vstupů a markantní rozdíly mezi jejími prvky.

Z těchto vlastností vyplývá, že bude třeba použít vhodné úpravy vzhledem k povaze zdrojového obrázku. S tím je spojena též chtěná nejednotnost chování programu, ten by neměl používat stejný postup pro každý obrázek, ale měl by uživateli umožnit zvolit a parametrizovat některou z nabízených metod.

3.1 Filtry

Základem všech úprav prováděných na zdrojovém obrázku je aplikace maticových, případně dalších obrazových filtrů. Je zde popsáno několik různých druhů filtrů, které mohou být při převodu obrázku použity. Nelze však tvrdit, že všechny z nich jsou využitelné či dokonce nezbytné – vývoj této části práce má experimentální charakter a mnoho metod, které se zprvu zdály vhodné, bylo později nahrazeno alternativami s lepšími výsledky. Finální varianty a kombinace filtrovacích prostředků byly zvoleny na základě empirických podkladů a analýzy konkrétního vzorku testovacích obrázků. Autor se tedy nesnaží nijak prokázat, že tyto metdy jsou pro účel definovaný zadáním nejvhodnější či dokonce jediné přípustné. Jejich vhodnost se vždy odvíjí od povahy testovaných obrázků a také hodnocení kvality výsledků je zcela subjektivním měřítkem. Autorovou snahou proto bylo zvolit takovou minimální množinu filtrů, která bude poskytovat co nejlepší výsledky pro co největší škálu obrázků.

3.1.1 Alpha kanál

První problematickou částí je přítomnost alpha kanálu, tedy průhledných oblastí, ve zdrojovém obrázku. Vzhledem k tomu, že ve výsledném obrázku není tato informace přípustná,

je třeba s ní nějak naložit.

Samotné ignorování této barevné složky však s sebou nese jistá rizika – pod průhlednou částí se může nacházet potencionálně jakákoli barva, což by mohlo fatálně odlišit vnímání obrazu počítačem od vnímání lidského. Často jsou hodnoty R, G, B složek v částech s úplnou průhledností nastaveny na hodnoty 0, tj. černá barva. Pokud má zbytek obrázku spíše světlejší nádech a je běžně vystavován na bílém pozadí, budou mít tyto černé skvrny vzniklé ignorací alpha kanálu zásadní vliv na jakoukoli další metodu založenou na porovnávání barevných složek.

Další z možností je uvažovat alphu jako další z kanálů, tedy rozšířit klasický tříbarevný prostor o další rozměr. Také tento přístup však vede k mnohým nesrovnalostem a nežádoucím efektům, jelikož alpha kanál se svým významem i vlastnostmi v mnoha směrech odlišuje od zbylých složek.

Nejlepším řešením se ukázalo upravit ostatní složky podle alpha kanálu. Uvažujme takto:

1. Výsledná barva musí být neprůhledná, tzn. alpha je na maximu.

2. Pokud je alpha kanál ve zdroji na své maximální hodnotě, tedy průhlednost je nulová, potom nemá žádný vliv na R, G, B složky výsledku.

3. Pokud je alpha kanál ve zdroji na své minimální hodnotě, což vyjadřuje úplnou průhlednost, potom R, G, B složky zdroje jsou bezvýznamné a výsledek přejímá zvolenou barvu pozadí.

4. Všchny ostatní případy jsou vyjádřeny lineárním přechodem mezi případy 2 a 3 v závislosti na hodnotě alpha kanálu ve zdroji.

Poslední neznámou je zmiňovaná barva pozadí. Za tu můžeme považovat barvu bílou, jelikož většinou nejlépe simuluje skutečnou barvu pozadí, na níž je obrázek vystavován a má nejmenší rušivé efekty. Obecně však lze zvolit libovolnou barvu. Vzorec pro převod barvy C obsahující alpha kanál do C' s hodnotami složek R, G, B, A v rozmezí $min \ldots max$ s barvou pozadí B splňující zmíněná kritéria je popsán ve vztazích 3.1. Graf převodní funkce pro jednu barevnou složku je znázorněn v obrázku 3.1.

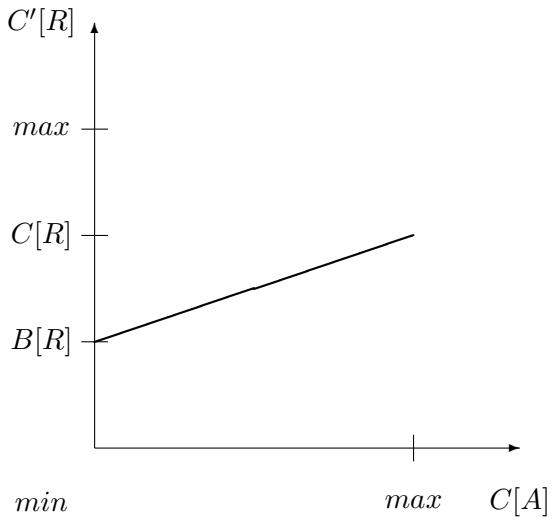
$$\begin{aligned} ratio &= (C[A] - min)/(max - min) \\ C'[A] &= max \\ C'[R,G,B] &= B[R,G,B] - ratio.(B[R,G,B] - C[R,G,B]) \end{aligned} \tag{3.1}$$

3.1.2 Převod do stupňů šedi

Jedná se o filtr, jehož vstupem je obecně jakýkoli obrázek I s body $p \in I$ o kterých můžeme říci jen to, že hodnoty jednotlivých složek $p[R, G, B]$ se mohou pro daný bod p vzájemně lišit. Výstupem je obrázek I' ve stupních šedi, pro který naopak platí $\forall p' \in I' : p'[R] = p'[G] = p'[B]$, lze tedy říci, že bod p' má jen jednu složku y, tedy odpovídající stupeň šedi. Převod $I \to I'$ tedy spočívá v nalezení takové transformace $p \to p'$, které bude odpovídat zmíněným podmínkám.

Jak popisuje J. C. Russ [13, s. 46], transformace může být provedena buďto omezením obrázku na jednu barevnou složku (R, G, B, případně H, S, I), vhodnější je však použití barevného filtru. Na výsledný obrázek mají poté vliv všechny barevné složky na vstupu,

Obrázek 3.1: Graf funkce potlačující alpha kanál



nikoliv jenom jedna. Transformační funkce je zpravidla ve tvaru $f : y = aR + bG + cB$, přičemž a, b, c reprezentují váhu původních složek.

Nejvěrohodnějších výsledků dosahuje filtr založený na YIQ/YUV barevném modelu [13, s. 40-41]. Složka Y tohoto modelu reprezentuje jas (*angl. luminance*) a určuje mj. stupeň šedi, který by byl použit při zobrazení v černobílé televizi. Její parametry reflektují lidské vnímání světlosti obrazu a je proto vhodným měřítkem pro převod barevného obrazu do stupňů šedi. Převod mezi RGB a YIQ modelem uvedený v [13, s. 40] je popsán v rovnicích 3.2, 3.3 a 3.4.

$$Y = 0.299R + 0.587G + 0.114B \tag{3.2}$$
$$I = 0.596R - 0.274G - 0.322B \tag{3.3}$$
$$Q = 0.211R - 0.523G + 0.312B \tag{3.4}$$

Při použití této varianty filtru je jako referenční brána právě složka Y a koeficienty uvedené u R, G, B v rovnici 3.2 jsou použity jako hodnoty a, b, c v uvedeném vztahu.

Intuitivně se zdá, že tento filtr by měl být první úpravou provedenou na zdrojovém obrázku. Obrázková křížovka ze své definice není barevným obrázkem a též většina filtrů popsaných v různých publikacích pracuje s černobílými obrázky. Při bližším zkoumání celého probému si však musíme uvědomit, že převodem do stupňů šedí se ztrácí z obrázku velké množství informací, což výrazně ovlivní jakékoli další upravy. Máme-li tedy v plánu na obrázek aplikovat více než jeden filtr, je výhodnější upravit ostatní filtry tak, aby akceptovaly všechny barevné složky, nežli hned na počátku použitím převodu do stupňů šedi degradovat informace nesené zdrojovým obrázkem.

Jako příklad uveďme detektor hran (popsaný v podkapitole 3.1.3). Ten hledá v obrázku sousedící pixely s rozdílnými hodnotami barevných složek. Mějme dva sousední pixely $p_1[R = 100, G = 100, B = 100]$ a $p_2[R = 100, G = 80, B = 200]$. Rozdíl v jednotlivých složkách mezi body p_1, p_2 je zřejmý, detektor hran by tedy tento přechod rozpoznal a do výsledného obrázku tuto informaci promítl. Pokud bychom však ještě před detekcí hran provedli převod do stupňů šedi výše popsaným způsobem s využitím složky Y ze vztahu 3.2,

nové body by měly podobu $p'_1[y = 100]$ a $p'_2[y = 100]$. Je zřejmé, že informace o přechodu se zcela ztratila a detektor nemá možnost původní hranu jakkoli rozpoznat.

I bez využití samotného převodu do stupňů šedi je však důležité si uvědomit význam a váhy jednotlivých barevných složek a jejich vliv na lidské vnímání, jak popisuje rovnice 3.2.

3.1.3 Detekce hran

Tento filtr je užitečný v mnoha případech, u složitějších obrázků je to často jediný způsob, jak dosáhnout alespoň nějakých výsledků. Má také výborné vlastnosti pro zpracovávání kreslených obrázků z animovaných seriálů, vyznačujících se výraznými konturami.

Cílem této úpravy je zvýraznění či extrakce hran v obrázku, tedy míst, ve kterých se prudce mění hodnoty jednotlivých barevných složek. Tyto plochy v ideálním případě odpovídají okrajům objektů v obrázku. Jednou z nejpoužívanějších metod je Sobelův detektor hran, respektive Sobelův operátor [4, s. 134-137].

Definujme černobílý obrázek dle [4, s. 1] jako funkci dvou proměnných $f(x, y)$, přičemž x, y jsou souřadnice nějakého bodu p v obrázku a hodnota $f(x, y)$ udává stupeň šedi bodu p. Jak je uvedeno v [4, s. 134-136], parciální derivací této funkce podle obou proměnných můžeme získat takzvaný vektor gradientu (rovnice 3.5), u nějž lze snadno určit jeho velikost (rovnice 3.6). Vysoké hodnoty této veličiny se nacházejí právě v místech hran, naopak v místech s konstantní barvou ($\delta f = 0$) je velikost gradientu nulová. Pro zrychlení výpočtu se často užívá zjednodušená, aproximační hodnota velikosti gradientu (rovnice 3.7).

$$\Delta f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\delta f}{\delta x} \\ \\ \frac{\delta f}{\delta y} \end{bmatrix} \tag{3.5}$$

$$|\Delta f| = \sqrt{G_x^2 + G_y^2} = \sqrt{\left(\frac{\delta f}{\delta x}\right)^2 + \left(\frac{\delta f}{\delta y}\right)^2} \tag{3.6}$$

$$|\Delta f| \approx |G_x| + |G_y| \tag{3.7}$$

Zbývá tedy jen určit hodnoty G_x, G_y. Ty je možno ze zdrojového obrázku získat použitím maticového filtru s již zmíněným Sobelovým operátorem. Jedná se o dvojici matic simulující derivace podle x a y. Tyto matice jsou uvedeny v rovnici 3.8.

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{3.8}$$

Popsaný algoritmus pracuje s černobílým obrázkem, avšak filtr lze snadno rozšířit pro použití na barevném obrázku. Detekce hran je poté prováděna pro každou barevnou složku zvlášť, tudíž výstupem je opět barevný obrázek, v němž každý z kanálů nese informaci o hranách v příslušné části barevného spektra.

3.1.4 Rozostření

Rozostřovací filtr (*angl. Blurring, Smoothing*) má za úkol snížit detail obrazu, rozmazat hrany a ostré části. Může též zmírnit poměr šumu v obraze. Jeho využitím v kombinaci

s vhodnou volbou prahu při tvorbě křížovky je možno dosáhnout zesílení hran a tmavých/světlých oblastí.

Nejčastěji užívaným principem pro docílení tohoto efektu je průměrování zkoumaného bodu se sousedními body (*angl. Neighborhood averaging* [13, s. 140]). Z matematického hlediska hraje ve filtru důležitou roli konvoluční matice W o rozměrech $m \times n$. Na jejích rozměrech a hodnotách závisí výsledný efekt. Ještě než popíšeme vliv jejích vlastností, podívejme se, jak probíhá výpočet nové hodnoty bodu. Předpokládáme definici obrázku provedenou v podsekci 3.1.3 jako funkci $f(x, y)$ souřadnic bodu. Funkci $f'(x, y)$ která je rozostřením obrázku definovaného funkcí $f(x, y)$ podle matice W s rozměry m, n popisuje rovnice 3.9 [4, s. 121].

$$f'(x,y) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} W[i,j].f(x+i\ -m/2, y+j-n/2)}{\sum_{i=0}^{m}\sum_{j=0}^{n} W[i,j]} \tag{3.9}$$

Podstatné jsou tyto parametry matice W:

1. Rozměry m, n jsou lichá čísla – zkoumaný bod se tedy nachází ve středu matice.
2. Matice může být čtvercová ($m = n$), obdélníková ($m \neq n$), či „kruhová“ (prvky mimo kruh jsou nulové).
3. Asymetrické matice se používají pro dosažení efektu pohybu (*angl. motion blur*).
4. Prvky matice mohou být shodné, tedy $\forall i, j : W[i, j] = W[0, 0]$.
5. Na rozměrech m, n a poměru váhy středového bodu vůči ostatním bodům závisí síla rozostření.

Pro naše potřeby budeme uvažovat použití čtvercové symetrické matice, jelikož nechceme zvýrazňovat ani jeden z rozměrů. Nejjednodušší variantou je poté použití matice W_1 z rovnice 3.10 [13, s. 141]. Ze vztahu 3.9 vyplývá, že hodnota nového bodu potom bude určena aritmetickým průměrem svého okolí.

$$W_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.10}$$

Chceme-li zvýraznit roli vzdálenosti bodů v okolí od zkoumaného bodu na výslednou hodnotu, můžeme použít matici W_2 z rovnice 3.11 [13, s. 142].

$$W_2 = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.11}$$

Ještě dále je s touto myšlenkou možno zajít použitím matic založených na Gaussově funkci [16]. Hodnoty prvků takové matice jsou udány Gaussovou funkcí pro vhodně zvolené parametry σ, α a μ. Rozměry matice jsou v závislosti na parametrech zvoleny tak, aby prvky mimo matici měly po zaokrouhlení nulovou hodnotu. Možnou podobu takovéto matice pro $\sigma = 0.391$ ukazuje rovnice 3.12 [13, s. 143].

$$W_3 = \begin{bmatrix} 1 & 4 & 1 \\ 4 & 16 & 4 \\ 1 & 4 & 1 \end{bmatrix} \tag{3.12}$$

3.1.5 Prahování

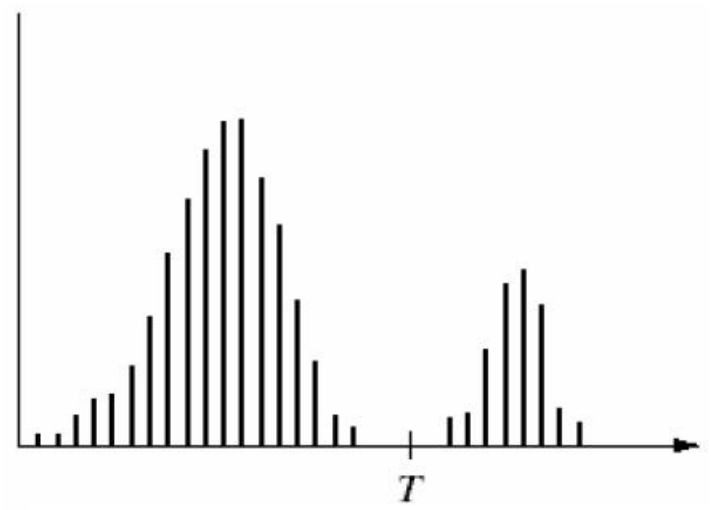
Prahování (*angl. thresholding*) je úprava často používaná při potřebě rozlišit objekty od pozadí obrázku. Výsledkem je binární obraz (reprezentovaný např. černými a bílými body). Právě tyto parametry splňuje též jakékoli řešení obrázkové křížovky.

Jádrem je rozdělení histogramu obrázku na 2 části pomocí zvolené hodnoty T, označované jako práh (*angl. threshold*). Výsledný obraz $f'(x, y)$ vzniklý aplikováním prahu T na zdrojový obrázek $f(x, y)$ popisuje vztah 3.13 [4, s. 596]

$$f'(x,y) = \begin{cases} 0 & \text{pro } f(x,y) \leq T \\ 1 & \text{pro } f(x,y) > T \end{cases} \tag{3.13}$$

Největším problémem tohoto filtru je správná volba prahu T. Snaha je vždy najít v histogramu „údolí“, podle nějž lze obrázek rozdělit, jak ukazuje obrázek 3.1.5.

Obrázek 3.2: Ukázka vhodné volby prahu T [4, s. 556]



Nejjednodušší možností je fixní volba prahu. Tento způsob je však použitelný pouze v případě, že předem známe povahu obrázku, jako např. u automatizované kontroly bezvadnosti výroby v sériích snímků produktů. Pro volbu odpovídajícího prahu obecného obrázku je však třeba využít některou z komplikovanějších metod.

Další možností je výběr průměrné hodnoty bodu v obrázku. Tato metoda dosahuje dobrých výsledků v obrázcích s vyrovnaným poměrem plochy objektu vůči ploše obrazu, pokud však jedna z těchto hodnot výrazně převažuje druhou, je možné, že algoritmus klasifikuje celý obrázek jako objekt, respektive pozadí.

Velmi dobrých výsledků dosahuje iterativní metoda výpočtu prahu T pro obrázek I je popsána algoritmem 6 [4, s. 599].

Problémem všech doposud zmíněných metod je fakt, že využívají jeden globální práh pro celý obrázek. To může vést ke zhoršení výsledků zejména u obrázků s velkými změnami v osvětlení. Tento nedostatek se snaží odstranit základní adaptivní prahování (*angl. Basic Adaptive Thresholding*, [4, s. 600]), jehož principem je rozdělení celého obrázku do několika podčástí, výpočtu a následné aplikace prahu pro každou tuto podčást zvlášť.

Algoritmus 6: Iterativní výpočet prahu T pro obrázek I

1. Zvol počáteční hodnotu prahu T.
2. Spočítej průměrnou hodnotu μ_0 všech bodů obrázku I klasifikovaných prahem T jako pozadí a průměrnou hodnotu μ_1 všech bodů klasifikovaných prahem T jako objekt.
3. Spočítej novou hodnotu $T = \frac{1}{2}(\mu_0 + \mu_1)$.
4. Pokud se nově spočítaný práh liší od předchozího, vrať se ke kroku 2.

Poněkud problematičtější je aplikace prahu na barevné obrázky. Jednou z možností je provést nejdříve převod obrázku do stupňů šedi a poté prahovat, jak však bylo ukázáno v podsekci 3.1.2, tento postup s sebou nese mnohá úskalí. Spočítat práh pro každou barevnou složku zvlášť není nic složitého. Otázkou však zůstává, jak provést porovnávání bodu p s prahem T, když se nejedná a čísla, nýbrž o n-tice čísel. V následujícím textu je popsáno jedno z možných řešení.

Uvažujme třísložkový práh $T[R, G, B]$ obrázku s třísložkovými body $p[R, G, B]$. Rozdíl $\Delta R = p[R] - T[R]$ je možno vnímat jako míru, se kterou červená složka přiřazuje bod p k objektu ($\Delta R \geq 0$), respektive k pozadí ($\Delta R < 0$). Analogicky je možno postupovat pro zbylé složky G, B. Dostaneme tak tři hodnoty, z nichž každá přiřazuje bod k pozadí či k objektu. Při konečné kategorizaci bodu lze tedy těchto hodnot využít. Musíme však brát v úvahu, že podle vzorce 3.2 vnímá lidské oko každou složku s jinou citlivostí. Výsledná funkce pro kategorizaci bodu p podle třísložkového barevného prahu T má tedy tvar popsaný vzorcem 3.14. Proměnná ΔC ve vzorci vyjadřuje celkovou míru udanou váhovým průměrem dílčích složek, se kterou je objekt přiřazován k objektu či pozadí.

$$\begin{aligned} \Delta R &= p[R] - T[R] \\ \Delta G &= p[G] - T[G] \\ \Delta B &= p[B] - T[B] \\ \Delta C &= 0.299\Delta R + 0.587\Delta G + 0.114\Delta B \\ p' &= \begin{cases} 0 & \text{pro } \Delta C < 0 \\ 1 & \text{pro } \Delta C \geq 0 \end{cases} \end{aligned} \tag{3.14}$$

3.1.6 Podvzorkování

Podvzorkování je úprava spočívající v převedení obrázku I o rozměrech $m \times n$ na obrázek I' o rozměrech $m' \times n'$, přičemž $m' < m, n' < n$. Zejména při velikých rozdílech mezi těmito veličinami se z obrázku ztrácí velké množství informací, je tedy vhodné provádět ji až jako poslední a velmi obezřetně. Vzhledem k tomu, že obrázkové křížovky mají obecně menší rozměry než obrázky, bude ve většině případů třeba této úpravy při převodu obrázku na křížovku využít.

Obecně lze říci, že při podvzorkování tvoříme z obdélníkové oblasti zdrojového obrázku, čítající několik bodů, jeden bod výsledného obrázku. Rozměr této oblasti může být vyjádřen jako $m/m' \times n/n'$.

Nejjednodušší a nejrychlejší metodou je vybrání jednoho bodu z této oblasti (např. prvního, prostředního či náhodného) a přiřazení jeho hodnoty výslednému bodu. Je zřejmé,

že tato metoda může lehce vytvořit mnoho chyb.

Poněkud komplikovanější je metoda založená na aritmetickém průměru bodů zdrojové oblasti, jenž je následně použit jako hodnota výsledného bodu. Problémem však může být zejména u obrazů s malou barevou hloubkou (binární obrazy) zaokrouhlování spočítaného průměru. Tuto nevýhodu můžeme vyřešit buď manuálním určením prahu, který se použije při zaokrouhlování, nebo distribucí vzniklé chyby do dalších částí výsledného obrázku.

Kapitola 4

Architektura programu

V této kapitole jsou popsány základní postupy použité při implementaci aplikace `picross`, která demonstruje teoretické poznatky definované v kapitolách 2 a 3. Budou diskutovány použité nástroje a metody, případně nabídnuty alternativy.

4.1 Vývojové prostředí, knihovny

Aplikace byla vyvíjena a testována v operačním systému Linux na počítači s dvoujádrovým procesorem Intel Core 2 Duo T5750 s frekvencí 2 GHz, grafickou kartou NVIDIA GeForce 9500M GS a 4 GB operační paměti DDR2 667 MHz.

Pro vývoj programu byl zvolen jazyk C++ a prostředí Qt verze 4.7, spravované firmou Nokia. Jedná se o komplexní toolkit šířený pod licencí GNU Lesser General Public License s řadou prostředků a knihoven pro tvorbu aplikací s uživatelským rozhraním, vlastním IDE a debuggerem (Qt Creator, viz [10]). K celému toolkitu je sepsána kvalitní dokumentace přístupná na webu [9], včetně zpětné dokumentace předchozích verzí.

Významnou vlastností tohoto prostředí je komunikace mezi objekty (potažmo widgety) pomocí tzv. signálu a slotů. Tento sofistikovaný přístup k výměně informací a šíření událostí umožňuje dynamický vznik a zánik vazeb mezi objekty za běhu programu a rozšiřuje tak běžné prostředky jazyka C++. K dosažení tohoto chování je však třeba používat kompilátor metaobjektů (*angl. Meta Object Compiler*) a speciální makra, což snižuje přenositelnost a znovupoužitelnost kódu mimo Qt.

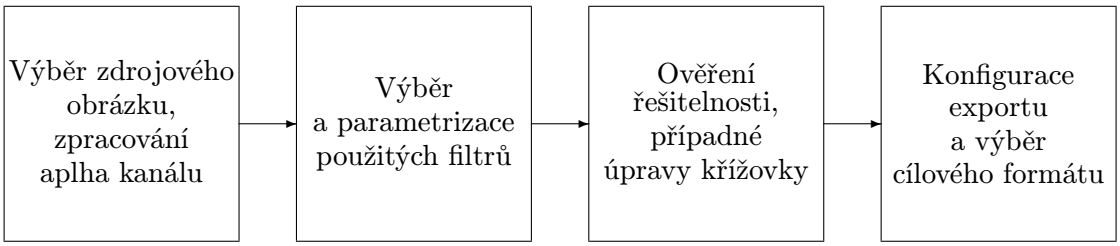
Přestože Qt nativně podporuje jazyk C++, je možné prostřednictvím různých vazeb (*angl. Bindings*) používat i jiné jazyky, například oblíbený python. Aplikace vyvíjené s tímto toolkitem je též možné exportovat na řadu platforem (Symbian, Linux, Mac OS atp.).

Pro účely zpracování obrazu bylo zprvu využíváno knihovny OpenCV [15]. Jedná se knihovnu šířenou pod licencí BSD obsahující řadu užitečných funkcí pro práci s obrázky a videem. Z důvodu jednodušší distribuce a především lepší kontroly nad prováděnými změnami byly později všechny použité funkce reimplementovány pomocí standardních prostředků Qt.

4.2 Návrh uživatelského rozhraní

Účelem uživatelského rozhraní je rychle, jednoduše a intuitivně provést uživatele tvorbou křížovky. Samotný proces je možno rozdělit do několika kroků, jak je znázorněno v diagramu

Obrázek 4.1: Jednoduché schéma aplikace



na obrázku 4.1. Rozhraní musí uživateli umožnit snadno se mezi těmito kroky pohybovat a dynamicky ovlivňovat nastavení s nimi související.

4.3 Implementace

Centrální komponenta aplikace s názvem `Widget` je implementována jako dceřiná třída třídy `QWidget` a jako hlavní prvek využívá komponentu `QTabWidget`. Jednotlivé panely poté představují dílčí kroky z obrázku 4.1 a jejich náplní jsou objekty tříd `stepX`, kde `X` je pořadí daného kroku (`1` až `4`). Všechny tyto třídy opět dědí z třídy `QWidget`.

Objekty `stepX` si předávají informace o svých změnách pomocí spojení signálů a slotů. Při tomto však nedojde k aktualizaci celého widgetu, objekt si jen zapamatuje nové vstupy. K překreslení na základě konkrétních vstupů dojde až při volání metody `revalidate`. Ta je prováděna vždy při úpravě nastavení zobrazeného kroku. Pro objekty v neaktivních panelech je volána až při jejich aktivaci, případně aktivaci některého z následujících kroků. Tato optimalizace zabraňuje tomu, aby se např. při opakovaných změnách zdrojového souboru bez opuštění prvního panelu bezúčelově přepočítávaly data ve zbytku aplikace. Zejména zpracování obrazu v kroku 2 totiž není nijak jednoduché a jeho opakované provádění by zbytečně zpomalovalo celou aplikaci.

Všechny použité obrazové filtry jsou implementovány v knihovně `filters.cpp`. Prvky zobrazující křížovku s různými vlastnostmi jsou objekty třídy `PicrossWidget`. Křížovka samotná je reprezentována třídou `Picross`.

4.3.1 Knihovna filtrů

Vybrané filtry z kapitoly 3 jsou implementovány v souboru `filters.cpp`. Všechny filtry pracují s objekty třídy `QImage`. Lze využít následující funkce:

- `getMeanThreshold` – vrátí spočítanou hodnotu prahu pro zadaný obrázek získanou jako aritmetický průměr.
- `getIterativeThreshold` – vrátí spočítanou hodnotu prahu pro zadaný obrázek získanou iterativní metodou.
- `threshold` – vyprahuje obrázek podle zadaného globálního prahu.
- `sobelEdgeDetection` – detekuje hrany v obrázku pomocí Sobelova operátoru.
- `grayscale` – převede obrázek do stupňů šedi použitím YIQ/YUV barevného modelu.

- `gaussianBlur` – vyhladí obrázek použitím Gaussovy matice.
- `suppressAlpha` – potlačí alpha kanál a nahradí jej zadanou barvou pozadí.
- `mergeImage` – zprůměruje dva obrázky se shodnými rozměry.

4.3.2 Třída `Picross`

Instance této třídy dědící z `QObject` reprezentuje konkrétní obrázkovou křížovku. Zapouzdřuje v sobě jak výsledek, tak i postup řešení křížovky. Za zmínku stojí metoda `solutinStep` a statická funkce `possibilities` implementující algoritmy z podsekce 2.5.3. Zajímavá je též metoda `update` zajišťující rozeslání aktualizačních signálů instancím třídy `PicrossWidget` napojeným na referencovanou křížovku.

4.3.3 Třída `PicrossWidget`

Jedná se o dceřinou třídu nadtřídy `QTableWidget`, jejímž úkolem je zobrazení výsledku, případně postupu řešení nějakého objektu třídy `Picross`. Umožňuje ovlivňovat řadu vlastností, jako například velikost polí, zobrazení mřížky atd.

Spojení s konkrétní křížovkou je zprostředkováno metodou `load`, která zajistí napojení příslušných slotů na signály vysílané křížovkou. Po provedení změn v křížovce tedy stačí zavolat na ni metodu `update` a všechny napojené instance třídy `PicrossWidget` si aktualizují svůj obsah v reakci na přijaté signály.

Při kliknutí na některé z polí emituje objekt signál `boxClicked`, čehož je využito při úpravě křížovky v kroku 3.

4.3.4 Krok 1

V prvním kroku je uživatel vyzván ke zvolení zdrojového obrázku a barvy, kterou bude nahrazen jeho alpha kanál.

Pro volbu obrázku z lokálního umístění je využita standardní funkce `QFileDialog::getOpenFileName`, případně je možno napsat cestu ručně do odpovídajícího pole. Manipulace s obrázkem je implementována s pomocí třídy `QImage`, z čehož vyplývá i seznam podporovaných formátů (viz [9]). Výběr barvy je zajištěn funkcí `QColorDialog::getColor`. Potlačení alpha kanálu pomocí zvolené barvy je implementováno na základě poznatků popsaných v podsekci 3.1.1.

V případě, že zvolený obrázek není možno zpracovat, je toto uživateli oznámeno a přístup do ostatních kroků je znemožněn. V opačném případě je uživateli zobrazen náhled a může pokračovat dále.

4.3.5 Krok 2

V druhém kroku si uživatel vybere jednu z nabízených sad filtrů, které lze na obrázek aplikovat. Má také možnost některé úpravy parametrizovat.

Uživatel volí mezi těmito variantami:

- *Neprovádět na obrázku žádnou úpravu.* Zejména černobílé a binární obrázky lze úspěšně převést na hezkou křížovku bez nutnosti aplikace jakýchkoliv dalších filtrů.

- *Sobelova detekce hran.* Pro tuto úpravu je využit algoritmus založený na Sobelově operátoru popsaný v podsekci 3.1.3. Jeho použití je velmi vhodné např. u postav z animovaných seriálů či velmi složitých obrazů.

- *Rozmazání obrazu.* Při této volbě je na zdrojový obrázek aplikován vyhlazovací filtr popsaný v podsekci 3.1.4. Za konvoluční matici je dosazena matice o rozměrech 5×5 s hodnotami předpočítanými podle Gaussovy funkce, jak je popsáno tamtéž. Tato možnost je obzvláště vhodná pro obrázky s velmi tenkými linkami.

Zdrojový obrázek je následně vyprahován použitím algoritmů popsaných v podsekci 3.1.5. Přesný postup spočívá v určení prahu iterativní metodou, při jejíž inicializaci je využit průměrný práh. Vypočítaná hodnota se poté použije jako globální práh. Adaptivní metoda se ve výsledné aplikaci nevyskytuje ze dvou důvodů. Tím prvním je skutečnost, že rozdíly jasu jednotlivých podčástí obrazu, které adaptivní prahování potlačuje, jsou v tomto případě žádoucí – pokud bychom obraz příliš hustě segmentovali, při konečném podvzorkování by měly jednotlivé oblasti velice podobné vlastnosti, což by ztěžovalo rozhodování o barvě výsledného bodu. Druhým důvodem je fakt, že použití globálního prahu pro celý obrázek umožňuje uživateli jeho konkrétní hodnotu jedoduše upravit podle potřeb, čehož je v aplikaci využito.

Užitečná je též možnost jednoduše invertovat hodnoty bodů ve vyprahovaném obrázku. Finální výsledek úprav je opět zobrazen uživateli jako náhled.

4.3.6 Krok 3

Ve třetím kroku se konečně dostaneme k vytváření samotné křížovky. Uživatel má možnost zvolit si výšku a šířku výsledné křížovky. Podle těchto parametrů je poté zdrojový binární obraz, vzniklý prahováním v kroku 2, převeden na křížovku. Hodnota pole křížovky je určena podle veličin s a r. Proměnná s označuje počet černých bodů ve zdrojovém obrázku v oblasti odpovídající zkoumanému bodu křížovky. Proměnná r je převodní koeficient, který může uživatel nastavit dle potřeby pro který platí $0 \leq r \leq s_{max}$, kde s_{max} je maximální hodnota, které může dosáhnout veličina s, konkrétně tedy celkový počet bodů zdrojové oblasti. Hodnota daného bodu křížovky je potom dána vztahem 4.1.

$$p' = \begin{cases} \text{černá} & \text{pro } s \geq r \\ \text{bílá} & \text{jinak} \end{cases} \tag{4.1}$$

Drobné detaily je možno upravit kliknutím na kterékoliv pole zobrazené křížovky, které tím změní svou barvu.

Následně má uživatel možnost nechat program, aby se pokusil křížovku vyřešit. Řešení se dá krokovat, pouštět a zastavovat dle potřeby. Vzhledem k náročnosti řešení a nutnosti zachovat odezvu rozhraní jsou procesy řešení a krokování vyvedeny do jiných vláken. Chování těchto vláken implementují třídy `Stepper` a `Solver`, dědící z `QThread`. `Solver` navíc disponuje možností řešení ještě před dokončením synchronně ukončit (volání asynchronní metody `terminate` by nebylo v tomto případě vhodné ani užitečné).

Po dokončení činnosti některého z vláken je zobrazena zpráva o vzniklé situaci, tedy zda-li je křížovka řešitelná či ne, případně že se ještě nepodařilo řešení dokončit.

4.3.7 Krok 4

V posledním kroku procesu již zbývá jen upravit výstup a exportovat výsledek do zvoleného formátu. Zde je plně využito flexibility třídy `PicrossWidget`. Uživatel si může upravit

velikost polí a písma, zobrazit či skrýt řešení, ohodnocení či mřížku.

Program poté nabízí export do několika formátů obrázku a do formátu PDF. Exportovaná data jsou získána pomocí funkce `QPixmap::grabWidget`, název cílového souboru pomocí `QFileDialog::getSaveFileName`. Po provedení exportu má uživatel možnost pokračovat ve vytváření dalších křížovek či úpravách té současné, nebo může program kdykoli ukončit.

Kapitola 5

Závěr

Cílem práce bylo vytvoření programu pro generování obrázkové křížovky na základě obecného obrázku. V jednotlivých kapitolách byla nastíněna úskalí tohoto úkolu a metody, kterými je lze překonat.

Přínos lze spatřovat na poli umělé inteligence a prohledávání stavového prostoru. Zejména obecně platná tvrzení dokázaná v kapitole 2 nabízejí cenné informace využitelné při sestavování vhodných řešících algoritmů a jejich optimalizaci.

Výzkumnou hodnotu v oblasti zpracování obrazu mají též závěry vyvozené z experimentů s obrazovými filtry popsanými v kapitole 3. Na jejich základě lze zvolit správný postup pro zpracování obecného obrazu za účelem vytvoření obrázkové křížovky případně jiného hlavolamu založeného na binárním rastrovém obrazu.

Program `picross` implementovaný jako součást této práce umožňuje uživateli jednoduše aplikovat popsané postupy a vytvořit si během chvíle kvalitní, vyluštitelnou obrázkovou křížovku.

Výsledky práce uspokojivě naplňují požadavky definované zadáním, zároveň však poskytují dostatečný prostor pro další rozšíření, případně alternativní přístupy ke zkoumané problematice. Možným pokračováním ve vývoji projektu je testování a parametrizace dalších obrazových filtrů a jejich sérií, zejména pro specifické skupiny zdrojových obrázků. Na práci lze navázat též zavedením víceřádkových metod řešení, případně implementací aplikace umožňující řešení obrázkových křížovek přímo v počítači.

Literatura

[1] Balcárek, L.: *Řešitel hry Griddlers*. Bakalářská práce, FIT VUT v Brně, Brno, 2010.
URL https://www.fit.vutbr.cz/study/DP/rpfile.php?id=10380

[2] Clay Mathematics Institute: *The Millenium Prize Problems*, 2011 [cit. 2011-05-09].
URL http://www.claymath.org/millennium/

[3] Dalgety, J.: NONOGRAM. 2010 [cit. 2011-05-09].
URL http://nonogram.co.uk/

[4] Gonzalez, R. C.; Woods, R. E.: *Digital Image Processing*. New Jersey: Prentice Hall, druhé vydání, 2002, ISBN 0-201-18075-8.

[5] Moura, L.: INTRODUCTION TO THE THEORY OF NP-COMPLETENESS. 2002 [cit. 2011-05-09].
URL http://www.site.uottawa.ca/~lucia/courses/4105-02/np.pdf

[6] Nagao, T.; Ueda, N.; Sato, C. P. T.; aj.: NP-completeness Results for NONOGRAM via Parsimonious Reductions. Technická zpráva, Department of Computer Science, Tokyo Institute of Technology, 1996.
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.5277&rep=rep1&type=pdf

[7] Nintendo: Picross 3D. 2010 [cit. 2011-05-09].
URL http://www.picross3d.com/

[8] Nintendo: Picross DS. 2010 [cit. 2011-05-09].
URL http://www.nintendo.co.uk/NOE/en_GB/games/nds/picross_ds_2915.html

[9] Nokia Corporation: Qt Reference Documentation. 2010 [cit. 2011-05-09].
URL http://doc.qt.nokia.com/4.7/index.html

[10] Nokia Corporation: Qt Creator IDE and tools. 2011 [cit. 2011-05-09].
URL http://qt.nokia.com/products/developer-tools/

[11] Olšák, P.; Olšák, M.: GRID: Program na řešení barevných i černobílých lušťovek. 2003 [cit. 2011-05-09].
URL http://www.olsak.net/grid.html

[12] Rusnak, M.: *Maľované krížovky*. Ročníkový projekt, Univerzita Komenského v Bratislave, Bratislava, 2008.
URL http://griddlers.wakeland.org/index.html

[13] Russ, J. C.: *The Image Processing Handbook*. CRC Press, Čtvrté vydání, 2002, ISBN 0-8493-1142-X.

[14] SILENTIUM: Pravidla pro řešení malovaných křížovek. *Malované křížovky – kódované obrázky*, ročník 7, č. 2, 2011: str. 42, ISSN 1335-9525.

[15] WWW stránky: OpenCV 2.0 C Reference. 2009 [cit. 2011-05-09].
URL http://opencv.willowgarage.com/documentation/index.html

[16] WWW stránky: Gaussova funkce – Wikipedie. 2011 [cit. 2011-05-09].
URL http://cs.wikipedia.org/wiki/Gaussova_funkce

[17] WWW stránky: Griddlers.net. 2011 [cit. 2011-05-09].
URL http://www.griddlers.net/

[18] WWW stránky: Průnik – Wikipedie. 2011 [cit. 2011-05-09].
URL http://cs.wikipedia.org/wiki/Pr%C5%AFnik

[19] Zbořil, F.; Zbořil, F.: Základy umělé inteligence IZU: Studijní opora. 2007.

Příloha A

Obsah CD

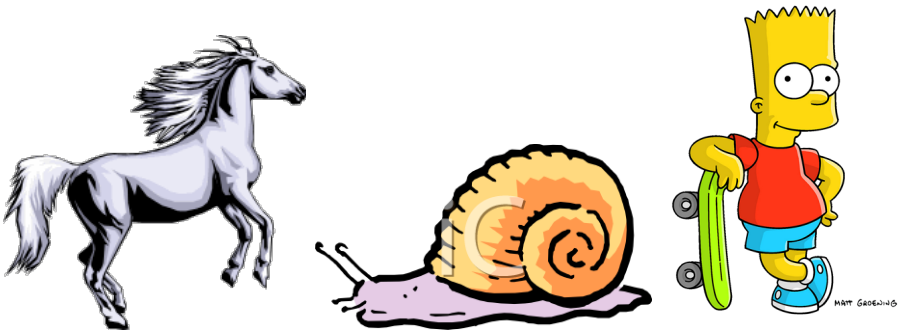
CD obsahuje tyto adresáře a soubory:

- `tex` – obsahuje zdrojové soubory v LaTeXu pro tuto práci.
- `code` – obsahuje zdrojový kód a projektový soubor nástroje `QtCreator` aplikace `picross`. Překlad lze provést sadou příkazů `qmake && make`.
- `source` – obsahuje několik zdrojových obrázků, které je možno využít jako vstup aplikace `picross`.
- `result` – obsahuje vzorek obrázkových křížovek vytvořených v aplikaci `picross`.
- `installLinux32` – obsahuje 32-bitovou verzi aplikace `picross` pro Linux. Spuštění skriptem `picross.sh`.
- `installLinux64` – obsahuje 64-bitovou verzi aplikace `picross` pro Linux. Spuštění skriptem `picross.sh`.
- `installWindows32` – obsahuje 32-bitovou verzi aplikace `picross` pro Windows. Spuštění pomocí `picross.exe`.
- `readme.txt` – soubor se základní nápovědou.
- `picross.pdf` – tato dokumentace.

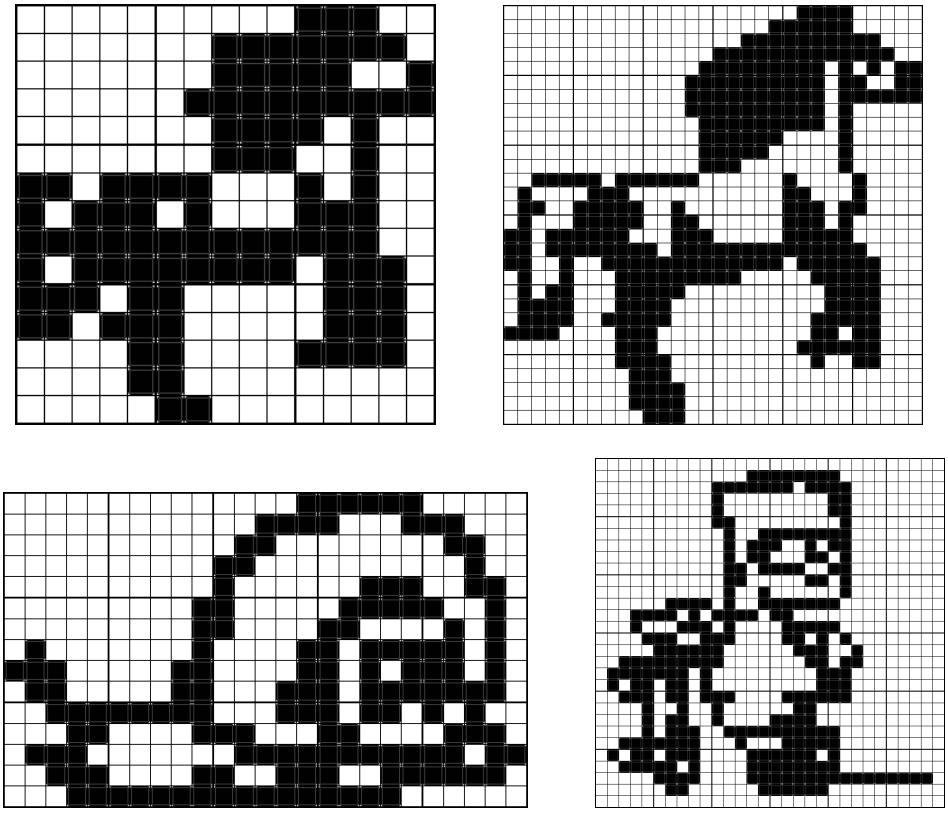
Příloha B

Příklady vstupu a výstupu

Obrázek B.1: Vstupy (zdroj: internet)



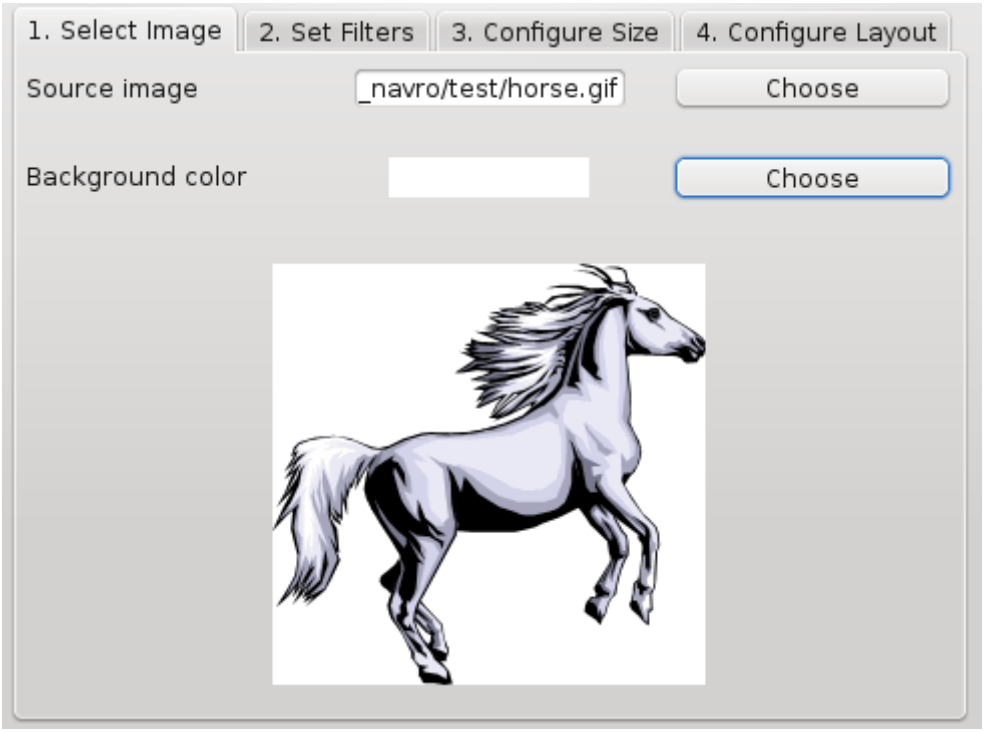
Obrázek B.2: Výstupy



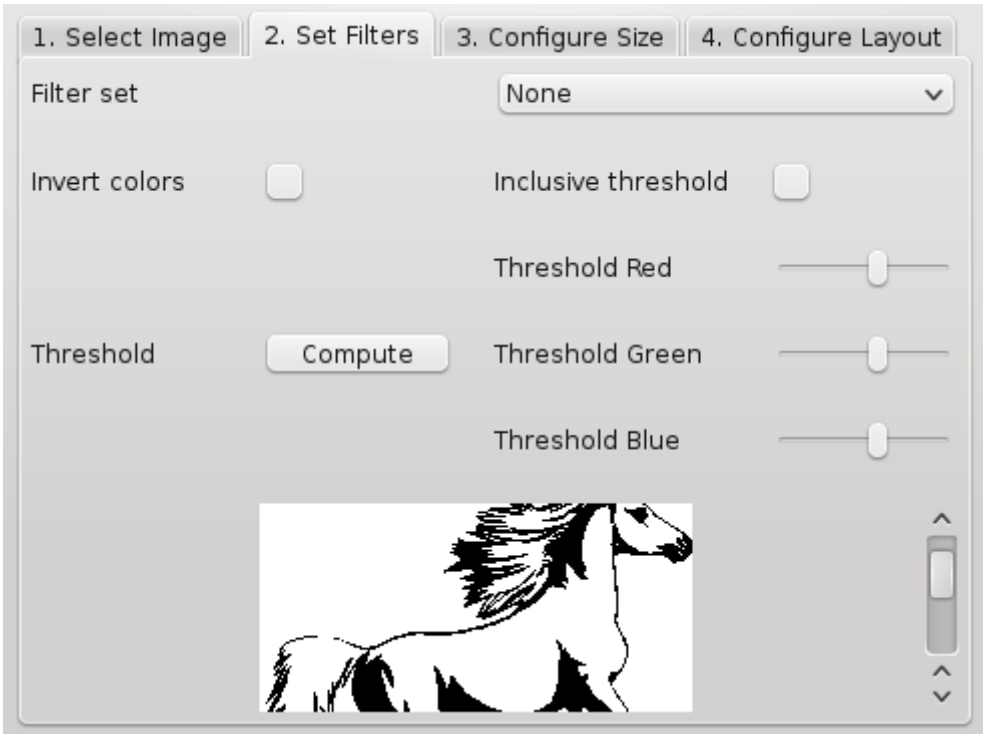
Příloha C

Ukázka aplikace

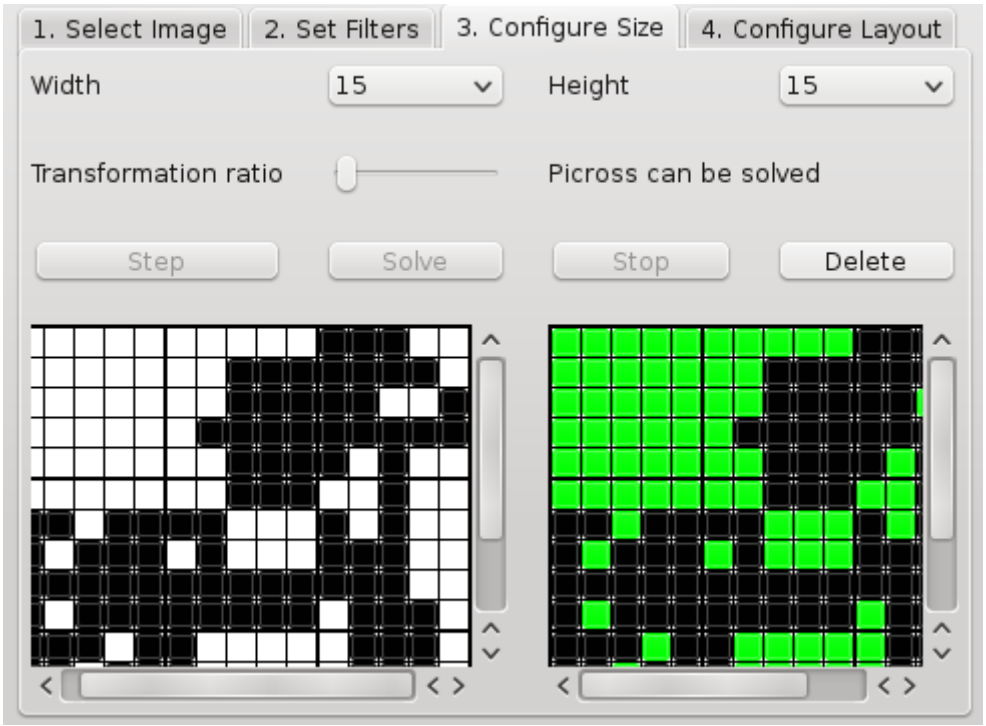
Obrázek C.1: Krok 1



Obrázek C.2: Krok 2



Obrázek C.3: Krok 3



Obrázek C.4: Krok 4